12th ICCRTS

Adapting C2 to the 21st Century

Use of a Systems Information Broker to Aide in the Dynamic Interfacing of C2 Nodes

Networks and Networking
C2 Metrics and Assessment
C2 Technologies and Systems

Dagohoy H. Anunciado [STUDENT]
Dagohoy H. Anunciado
SPAWARSYSCEN 24226, San Diego, CA & Naval Postgraduate School, Monterey CA
53605 Hull St Bldg A-33
San Diego CA 92152-5001
619-553-5604/619-553-6025 FAX
doug.anunciado@navy.mil

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2007		**00-00-2007 to 00-00-2007**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Use of a Systems Information Broker to Aide in the Dynamic Interfacing of C2 Nodes	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Space & Naval Warfare Systems Center,53605 Hull St Bldg A-33,San Diego,CA,92152-5001**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
Twelfth International Command and Control Research and Technology Symposium (12th ICCRTS), 19-21 June 2007, Newport, RI

14. ABSTRACT

Missions assigned to military forces will change as world events occur. Recent events like the Indian Ocean Tsunami and Hurricane Katrina in the United States required a massive humanitarian effort that included military forces. Information about the event needed gathering, distributing, and analyzing to determine how best to use resources to help the people in the devastation. Once observers gather information, establishing communications is needed before information can be distributed. Command and Control (C2) node functions perform one or all of the tasks of information gathering, distribution, analysis, decision making, and distribution of decisions. C2 nodes in these situations will be mobile or fixed and will come and go as a mission unfolds. Interfacing of C2 nodes may be hampered when the interface mechanisms are not worked out before an event and would take time to manually work out, which delays rescue and relief efforts. This research defines a framework and methodology for dynamically interfacing C2 nodes to a C2 enterprise to accomplish large missions such as responding to operations other than war (OOTW), e.g., natural and man-made disasters, peacekeeping, and counter drug operations.1 Regional conflicts and general war are other situations requiring C2 enterprise to accomplish a large mission.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**33**	

Abstract

Missions assigned to military forces will change as world events occur. Recent events like the Indian Ocean Tsunami and Hurricane Katrina in the United States required a massive humanitarian effort that included military forces. Information about the event needed gathering, distributing, and analyzing to determine how best to use resources to help the people in the devastation. Once observers gather information, establishing communications is needed before information can be distributed. Command and Control (C2) node functions perform one or all of the tasks of information gathering, distribution, analysis, decision making, and distribution of decisions. C2 nodes in these situations will be mobile or fixed and will come and go as a mission unfolds. Interfacing of C2 nodes may be hampered when the interface mechanisms are not worked out before an event and would take time to manually work out, which delays rescue and relief efforts. This research defines a framework and methodology for dynamically interfacing C2 nodes to a C2 enterprise to accomplish large missions such as responding to operations other than war (OOTW), e.g., natural and man-made disasters, peacekeeping, and counter drug operations.[1] Regional conflicts and general war are other situations requiring C2 enterprise to accomplish a large mission.

1. Introduction (Motivation)

Missions assigned to military forces will change as world events occur. Events like the Indian Ocean Tsunami[2] devastated coastal regions of Indonesia, Sri Lanka, India, and Thailand, and also affected Somalia, Myanmar, the Maldives, Malaysia, Tanzania, Seychelles, Bangladesh, South Africa, Yemen, Kenya, and Madagascar. Hurricane Katrina in the United States created a storm surge that caused severe and catastrophic damage along the Gulf coast, devastating the cities of Mobile, Alabama, Waveland, Biloxi, and Gulfport in Mississippi, and New Orleans and other towns in Louisiana. Levees separating Lake Pontchartrain from New Orleans were breached by the surge, ultimately flooding 80% of the city and many areas of neighboring parishes for weeks.[3] Both events required a massive humanitarian effort that included military forces.

When these types of events occur, information about the event needs gathering, distributing, and analyzing to determine how best to use resources to help the people in the devastation. Once observers gather information, establishing communications is needed before information can be distributed. Command and Control (C2) node functions perform one or all of the tasks of information gathering, distribution, analysis, decision making, and distribution of decisions. C2 nodes in these situations will be mobile or fixed and will come and go as a mission unfolds. Since C2 systems may have pieces of information that a decision maker will need, interfacing with other C2 systems is necessary in order get a big picture for decision makers to formulate their decisions. The goal of this research is to present and implement a systematic method for the dynamic interfacing of C2 systems. The core of this method is an entity called the Systems Information Broker (SIB). The SIB serves as an arbitrator that will determine whether the interfacing is feasible, and as a uniform interfacing platform to support the interfacing of real-time and non-real-time systems. To aid in the interfacing feasibility, a pre-formulated set of methods will be determined that are predicted to yield interfaces among systems. The focus of this research is determining the constraints on the methods used in interfacing systems that will allow a static calculation that can predict that an interface between systems is feasible.

2. Basic Architecture and Framework

The goal of this research is to present and prototype a systematic method to facilitate the dynamic interfacing of real-time and non-real-time systems. The core of the method is an entity called the Systems Information Broker (SIB). We propose to breakdown the responsibility of the SIB into two parts with this research focusing on the first part.

1) SIB will serve as an arbitrator that will determine whether the interfacing is feasible by considering the satisfaction of timing constraints and resource usage.
    1.1) SIB serves as an arbitrator taking into account the reconfiguring issues involved in the enterprise of systems supporting forces and units used to fulfill a new mission. We will need to create a calculation or mechanism for determining whether the proposed methods for interfacing systems are schedulable as systems are dynamically added, deleted, and immigrated.
    1.2) In addition, we will need to determine the optimality goals and constraints for the resources used on the interfacing systems. Before determining the optimality goals and

constraints, we will need to determine the metrics and calculation mechanisms that determine the current resource use. Once current resource use is known, one can then chose optimality goals and constraints for the interfaced resources. A mechanism will be needed to adjust the resource use to comply with the goals and constraints when resources are being over used. The SIB will use these mechanisms to determine if resources are used properly and make adjustments to comply with goals and constraints, but since we will not have complete information of the global state of the interfaced systems these mechanisms will only give suboptimal resource use. The goal is to still provide effective use of resource, but not necessarily optimal resource use.

2) SIB will also serve as a uniform interfacing platform to handle data interoperability and timing constraints to support the interfacing among systems. SIB will be used in an operational mode and serves as a uniform platform to handle the scheduling of system interactions, and interoperability among systems.

Figure 1 is the framework of the systematic method to facilitate the dynamic interfacing of real-time and non-real-time systems.

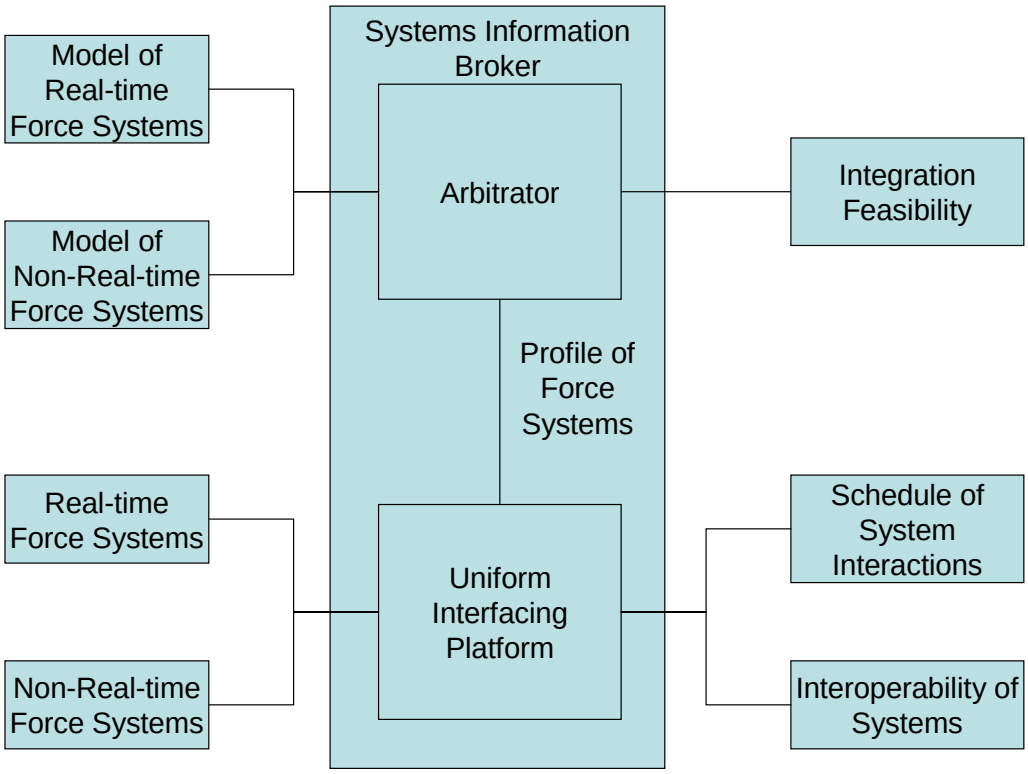


Figure 1 Framework of the proposed method

3. Mechanism for Determining if Systems Can Interface

We plan to develop a criterion that the SIB uses to determine whether interfacing systems is feasible. The SIB will use this criterion to determine if the enterprise of systems is schedulable after some systems are added, deleted, or immigrated from the enterprise. We imagine that this schedulability determination will be similar to embedded real-time schedulability but with a higher magnitude of timing constraint values due to network delays and jitter. The immigrating capability will be limited to non-real-time systems. To develop this criterion, a dynamic scheduling analysis algorithm for distributed systems with non-real-time and real-time tasks needs to be presented. Breaking down the mission the enterprise is required to accomplish into phases of operation and placing systems to address one phase may provide a way to bound the timing constraints that the systems handling a phase needs to meet. Calculating probabilities of moving from one phase to another may provide an additional means of bounding the timing constraints among real-time and non-real-time systems.

Another criterion that the SIB uses is to determine whether the enterprise of systems still has good resource usage after some systems are added, deleted, or immigrated from the enterprise. To develop this criterion, a resource usage metric needs to be defined and a method will be developed to compute this metric.

This framework will model systems with the base component being a single system. The model will not go much below the system level. A description of a system will include characteristics of the system and applications running on the system.

3.1. Alternative Methods for Interfacing Systems

The idea is to have several methods to choose from when interfacing systems. Methods will be ranked by scoring criteria that is explained in the next section, with the top scoring method being the primary interfacing method and the remaining methods as alternatives.

3.2. Modeling Systems Being Interfaced

We are working the model at a systems level where simple constructs and events are passed between systems. Modeling the systems with layers, and allow different layers of a system with past-through channels may allow for response times for the overall system not being hindered by individual layer transformations.

To create time-budgets with existing deployed applications and services we are going to need tools to measure resource usage using existing OS facilities. We are also going to need a method for determining excess resource capacity and heuristics to estimate it.

The excess capacity would be available to support the interfacing with other systems and few tasks or ultimately all tasks required to fulfill a mission thread.

Figure 2 illustrates the constructs used to interface systems together.

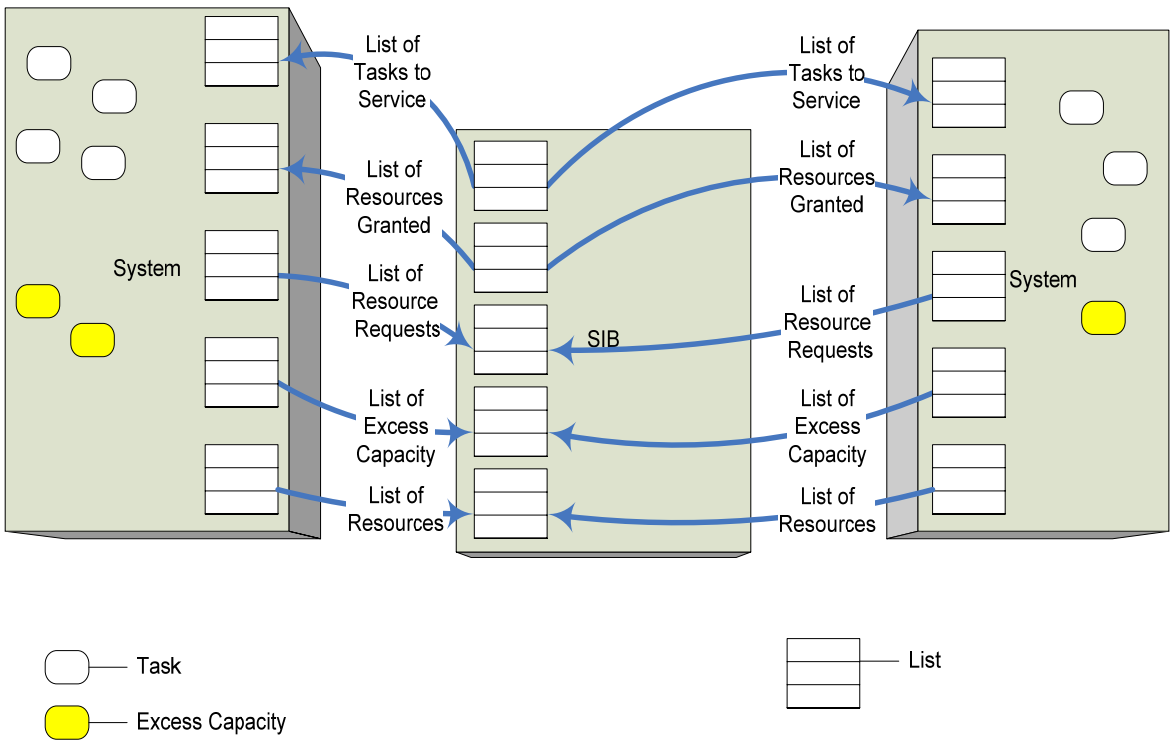


Figure 2 Modeling Constructs Used to Interface Systems

Systems would list out the resources it possesses and also list the resources it seeks. The systems send these lists to the SIB for recording and future servicing. Lists of excess resource capabilities would also be passed to the SIB. The SIB would coordinate the various lists of requested resources and map these with available resources. The goal is to do this mapping without human intervention, and then proceed to allocate resources to requests and then send back the lists of resource granted plus the list of tasks currently needing service.

3.3. Application Complexity Rating

Applications used on a system vary in degrees of complexity from simple to very complex. On the complex end are applications that require many data sets, perform a high number of calculations on a subset of the data sets, and graphically render the calculated results. Without these data sets, desired results may not be precise enough to be useful. Many times developers will have an idea of what data their applications need to ingest, but these data sets are not well documented such as what the data is making up the data set, the way data is being gathered, what organization is maintaining the data sets, or how to get access to the data sets.

Table 1 contains the attributes that can determine the complexity of an application. An application that is real-time and performs any of the other attributes would be considered a very

complex application. An application that is non-real-time, calculation intensive, data intensive, and graphical, or has only the real-time attribute is a complex application. A semi-complex application would be non-real-time and has two of the three remaining attributes. A simple application would be non-real-time and has only one of the remaining attributes.

Table 1 Complexity Attributes for an Application

Complexity	**Real-time**	**Calculation Intensive**	**Data Intensive**	**Graphical**
	Timing constraints bounding computed results	Requiring high number of math calculations	Requiring high number of data points	Requiring high use of graphics to rendering information
Simple				X
Simple			X	
Simple		X		
Semi-Complex			X	X
Semi-Complex		X		X
Semi-Complex		X	X	
Complex		X	X	X
Complex	X			
Very Complex	X			X
Very Complex	X		X	
Very Complex	X		X	X
Very Complex	X	X		
Very Complex	X	X		X
Very Complex	X	X	X	
Very Complex	X	X	X	X

3.4. Data Set Considerations

Without knowing what data sets or information one needs or an application, process, or system needs, one cannot perform processing with the data sets in order to get results from a formula or model. The terms process and task will be used the synonymously throughout this paper. Before processing formulas or models, data is needed and depending on the use of the formulas or models, continuous processing of formulas or models may also require continuous access to the data sets for results to be relevant to a user. Relevant results mean that a user will be able to take actions to avoid harmful consequences.

Understanding what data sets are used is a good starting point to keeping an application relevant to its users. Keeping the data sets organized, and knowing who and where to get updated data sets also adds to an application's relevance.

Making the data sets organized and simple to understand may make the maintenance of data sets easier. Plus a systems information base used to keep an enterprise of systems working may perform better with well organized data sets Furthermore, having a flat organization for the data sets may be an ideal way to understand the data an application or system will need.

3.5. Real-time and Non-real-time System Attributes

We will model systems by first separating real-time and non-real-time systems by their attributes. Table 2 contains the attributes used to model real-time systems used by a force.

Table 2 Real-time System Attributes

Attribute	Comment
Tasks	
CPU Cycles used	CPU cycles used to accomplish a tasks
Network resource usage	
Time Constraints: Finish Within or Maximum Execution Time	
Periodic, Event Driven, or Both for Task Execution	

Non-real-time force systems will be modeled with similar attributes shown in Table 3.

Table 3 Non-real-time System Attributes

Attribute	Comment
Tasks	
CPU Cycles used	CPU cycles used to accomplish a tasks
Network resource usage	
Periodic, Event Driven, or Both for Task Execution	

3.6. Breaking Down the System Resources

Resources used by systems will be broken down into system resources and network resources. Systems resources are further decomposed into CPU resources, memory resources, and I/O resources. Tied with each resource are resource concerns that have the potential of diminishing the quality of service of the resource.

Table 4 and Table 5 below have the resources being modeled plus the resource concerns for each resource.

Table 4 Real-time System Resources Modeled

Resource	Resource Concern
System	
CPU	Lack of CPU cycles to complete a task calculation that will cause a task to miss its deadline.
Memory	Lack of memory causing a task to miss its deadline.
I/O	Waiting for I/O resources that causes a task to miss its deadline.
Network	
Bandwidth	Lack of Bandwidth that causes a task to miss its deadline.
Quality of Service	Jitter and Latency that degrade information flow and causes a task to miss its deadline.

Table 5 Non-real-time Systems Resources Modeled

Resource	Resource Concern
System	
CPU	Lack of CPU cycles prevents a task from completing its computations in a usefully timeframe, i.e., meeting the mission thread time-budget.
Memory	Lack of memory prevents a task from completing its computations in a usefully timeframe, i.e., meeting the mission thread time-budget.
I/O	Waiting for I/O resources prevents a task from completing its computations in a usefully timeframe, i.e., meeting the mission thread time-budget.
Network	
Bandwidth	Lack of Bandwidth case prevents a task from completing its computations in a usefully timeframe, i.e., meeting the mission thread time-budget.
Quality of Service	Jitter and Latency that degrade information flow and prevents a task from completing its computations in a usefully timeframe, i.e., meeting the mission thread time-budget.

3.7. Scoring Criteria of Interfacing Methods

Interfacing methods are scored using criteria of the interfacing latency, capacity, and quality of service which includes availability and reliability. Other criteria may include cost of using the interface.

3.7.1. Computational Model for an Enterprise of Systems

Y. Qiao, et al., developed an admission control method for dynamic software reconfiguration in the systems of embedded systems (SoES) domain.[4] This method prevents dynamic software reconfiguration from damaging the high confidence of SoES. We plan use many of the concepts of the SoES admission control method to provide the computational model for the enterprise of systems (EoS) admission control method.

The EoS admission control method has two parts 1) modeling the systems making up the enterprise of systems and 2) the dynamic scheduling analysis for the EoS. The modeling of the systems mathematically describes the functional and non-functional aspects of the EoS requirements. This description has an external view model and an internal view model. The external view model is customer view focused, while the internal view model is designer view focused.

Appendix A describes the admission control model for SoES developed by Y. Qiao, et al., that we will modify and use in representing the EoS.

3.7.2. Maintaining System Schedule

Interfacing a system into an EoS must not interfere with an individual system's processing to meet local task schedules. An EoS schedule is first determined by the time-budget for the mission threads that an EoS supports. Development of time-budget allocations for time-critical mission threads is a recommendation of the Committee on C4ISR for Future Naval Strike Groups.[5]

The below tuple represents a mission thread time-budget.

$$(\text{MTID}, \text{TCM}, \text{MTD}, \text{R}) \quad (1)$$

MTID Mission Thread Identification

TCM Time to Complete Mission Thread

MTD Mission Thread Description

R Set of n Resources Needed to Accomplish Mission Thread, where $n > 0$.

A mission thread may be made up of multiple tasks and each task is represented by the following tuple.

$$(\text{TID}, \text{TCT}, \text{TD}, \text{TR}) \quad (2)$$

TID Task Identification

TCT Time to Complete Task

MTD Task Description

TR Set of Resources Needed to Accomplish Task where $\text{TR} \subseteq \text{R}$.

TCT is constrained by the mission tread time-budget, TCM. Resources needed to complete a task mainly include data sets, but may include computational, storage, networking, radio, and other physical resources.

3.7.3. Maintaining Optimality Goals and Constraints with Respect to Resource Usage

The goal for individual system resources usage is to utilize the resources close to maximum, at least 80%, with 20% to surge processing. But the emphasis is to maintain the system processing schedule even at the cost of underutilizing the resources.

3.7.4. Pulling it Together

The application complexity rating, data set considerations, real-time system attributes, and breaking down of resources will compose the majority of the internal view mentioned in Appendix A.

Once a request for interfacing a new system into the EoS occurs, the information specified above would need to be provided by the requesting system. A precedence graph would be built for the proposed new system in the EoS, breaking down the mission thread this new system will help to fulfill, into the constituent tasks as with precedence graph for a SoES in Appendix B.2. The mission tread precedence graph will also have non-real-time dependencies that when broken down into its constituent tasks will have non-real-time task interspersed throughout the precedence graph. These non-real-time tasks can act as separators between real-time segments in the precedence graph. For example, in Figure 3, task T_4 separates task T_7 and partially separates T_6 from the rest of the real-time tasks in the precedence graph.

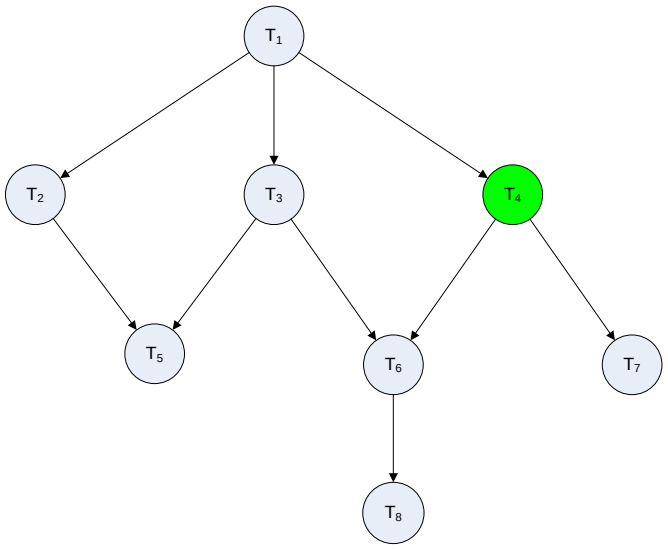


Figure 3 Precedence Graph with a Non-Real-Time Tasks T_4 Example

Since non-real-time tasks would act as separators in the precedence graph, the non-real-time computation needs to be handled separately using a mechanism to allocate a percentage of a system's CPU to best effort tasks processing. For example, initially setting aside 10% of a system's CPU for best effort processing. Profiling of the non-real-time tasks at the initial CPU allocation would provide an average maximum execution time, ready time, start time, and task interaction latency, which is beyond the scope of this paper. These average task attributes would than be used to seed the real-time task attributes described in Appendix B.2 for the non-real-time tasks. Thus making the non-real-time tasks pseudo real-time tasks and allow us to use the dynamic scheduling analysis presented in Appendix B for the proposed EoS.

If the analysis shows that a feasible schedule is achieved, then interfacing of the new system into the EoS is allowed. Otherwise the request to interface is denied and the denied request could then be sent to the "training process" to determine what settings could be used to obtain a feasible schedule. The "training process" would help determine options for the real-time time tasks in finding constraints that would lead to a feasible schedule or if the schedule is not being met due to the non-real-time tasks not completing. If non-real-time tasks are the reason for no feasible schedules, the "training process" could determine if allocating additional CPU to best effort tasks processing would provide a feasible schedule.

4. Conclusion

This research provides a systematic method for dynamically interfacing systems to form an enterprise of systems. At the EoS level the SIB serves as an arbitrator for the tasks requiring remote resources or remote execution on other systems in the EoS. Given the attributes of the systems and remote resource needed by a task, the SIB will determine if the tasks and the systems making up a mission thread will be able to interface with the EoS and continue to maintain local tasks schedules.

Appendix A Admission Control Method Modified for the Enterprise of Systems (EoS)

The functional and non-functional aspects of the EoS are represented with two views: the external view model and the internal view model.

The external view model is denoted as ζ', and represented as

$$\zeta' = (G, H) \tag{A1}$$

G is the functional emergent property vector that represents the functional aspect of the EoS requirements, $G = (g_1, g_2, \ldots, g_i)$, where $g_i (i \in [1, l])$. g_i denotes one of the functional emergent properties describing the emergent behavior of the EoS and l is the number of functional emergent properties. The most typical functional emergent property identified in the external view model is timing properties such as maximum response time.

H denotes non-functional emergent properties related to high confidence. It is described by a high-confidence metric vector, $H = (h_1, h_2, \ldots, h_i)$, where $h_i (i \in [1, z])$ is a high confidence metric and z is the number of high confidence metrics. Some typical metrics are failure rate, maximum time between two successive failures, the number of faults that can be tolerated, maximum time between safety violations and security level etc.

The Internal view model is denoted as ζ, and represented as

$$\zeta = (S, E, C, D, F_1, F_2) \tag{A2}$$

S is a component system set, $S = \{s_i \mid i \in [1, n]\}$. s_i denotes a component system constituting the EoS and n is the number of component systems in the whole EoS. E denotes the interaction sets between component systems, $E = \{e_{jk} \mid j, k \in [1, n]\}$, where e_{jk} denotes a set of interactions from component system s_j to component system s_k. C denotes the constraint sets on how the component systems are used in the given environment, $C = \{c_i \mid i \in [1, n]\}$. c_i is a set of constraints imposed on s_i. D denotes the constraint sets on interactions between component systems, $D = \{d_{jk} \mid j, k \in [1, n]\}$, where d_{jk} is a set of constraints applied to interactions in e_{jk}.

F_1 and F_1 are two mappings that refine emergent properties of EoS into local constraint sets imposed on component systems and interactions, i.e., $C = F_1(G,H)$ and $D = F_2(G,H)$.

In internal view model, timing constraints are included in C and D. Typical timing constraints include deadline and maximum execution time of the component system and latency of the interaction between two specific component systems. Furthermore, some resource constraints such as access mode and control constraints such as trigger method are also included in C and D. All these constraints can be extracted as parameters used by dynamic scheduling analysis for the EoS. In addition, each component system is either atomic or composite in internal view model. For convenience, to support the scheduling analysis, we take each atomic component system as a task to be scheduled by the scheduling algorithm.

Appendix B Dynamic Scheduling Analysis for an Enterprise of Systems

B.1 Dynamic Scheduling Task Model

Since EoS are characterized by dynamic combinations of component systems, in this paper we mainly consider the aperiodic tasks. Y. Qiao et al., presented a task model for the use of dynamic scheduling analysis in SoES and we propose to extend this model for the EoS dynamic scheduling analysis as follows:[6]

1) Each task T is described as a tuple $(a_T, r_T, D_T, v_T, E_T)$. Here, a_T is task T's arrival time and r_T is T's ready time. D_T denotes T's deadline. v_T is the number of T's different logic versions. E_T represents T's maximum execution time. For hard real-time tasks, E_T is a vector denoted by $(e_T^1, e_T^2, ..., e_T^m)$, where $e_T^j (j = 1,...,m)$ is the maximum execution time of task when it executes on processor p_j and m is the number of processors in the EoS. For soft real-time tasks, the maximum execution time E_T is a matrix denoted by $e_T^{ij} (i = 1,...,v_T; j = 1,...,m)$, where e_T^{ij} is the maximum execution time of task T's logic version i when it executes on processor p_j. Furthermore, for each $j (j = 1,...,m)$, the maximum execution time of each logic version is ordered such that $e_T^{1j} \le e_T^{2j} \le ... \le e_T^{v_T j}$. Non-real-time or best-effort tasks provided periods of execution time that can change at runtime.
2) Hard and software real-time tasks are non-preemptive and non-periodic, and these tasks can not be parallelized.
3) Besides processors, tasks might need some other resources such as data structures, variables, and communication buffers for their executions. Every task can access a resource either in shared mode or in exclusive mode.

The characteristics of the task listed above can be extracted from internal view computational model addressed in Appendix A. For example, the deadline and maximum execution time of a task can be derived from constraint sets imposed on the corresponding component system, and the access mode of a task can be derived in the same way.

B.2 Dynamic Scheduling Precedence Graph

We will use the same precedence graph concept as Y. Qiao, et al.[7], to represent tasks, but we will have two levels of representations. The first level will model task precedence at the individual system level, and the second level will model task precedence at the EoS level. Tasks requiring remote resources or remote execution on other systems will have their resources allocated and dispatched through the SIB.

Since the interaction of component systems in an EoS may be just as ubiquitous as SoES, so precedence constraints shall be considered during the scheduling analysis. These constraints specify whether a task needs to precede another task. If the output of task T_x is needed as input by task T_y, then task T_y is constrained to be preceded by task T_x. Furthermore, there are two kinds of precedence constraints: one is the AND constraint; another is the OR constraint. Accordingly, there are two types of tasks: the AND tasks cannot begin their computing until all their preceding tasks have completed, while the OR tasks can begin after any one of their preceding tasks complete.

Definition B.2.1: $T_x \prec T_y$ represents that task T_x must precede task T_y.

Definition B.2.2: The precedent-task set of task T_x is denoted by $\prec(T_x)$; that is, $\prec(T_x)$ indicates which tasks must be completed before T_x can begin.

Definition B.2.3: Assume T_x is a task. $T_x.readytime$ denotes its ready time; $T_x.met$ denotes its maximum execution time; $T_x.starttime$ denotes its start time.

Definition B.2.4: Assume I_{xy} is the interaction between task T_x and task T_y. $I_{xy}.lat$ denotes its latency.

The precedence constraint can be represented by means of a precedence graph. In a precedence graph, each node represents a task (or a component system). The arrows indicate which task has precedence over another task. Figure 4 shows an example of a precedence graph that we use in this paper.

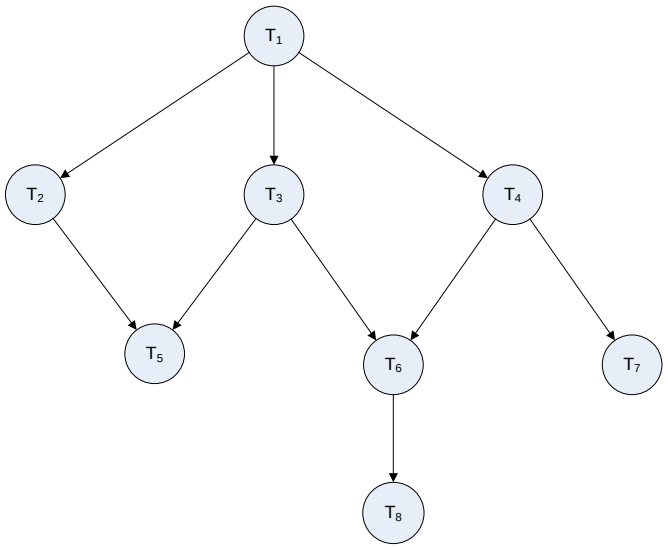


Figure 4 Precedence Graph Example

The precedent-task sets for the graph in Figure 4 are as follows:

$$\prec(T_1) = \varnothing$$
$$\prec(T_2) = \{T_1\}$$
$$\prec(T_3) = \{T_1\}$$
$$\prec(T_4) = \{T_1\}$$
$$\prec(T_5) = \{T_2, T_3\}$$
$$\prec(T_6) = \{T_3, T_4\}$$
$$\prec(T_7) = \{T_4\}$$
$$\prec(T_8) = \{T_6\}$$

The precedence graph can be constructed based on the internal view computational model since precedence relationships between tasks can be derived from interactions between component systems described in this model. In this case, AND and OR tasks can be identified by a control constraint, i.e., trigger method imposed on each component system, which is described in the internal view computational model. If the trigger method of a component system is *trigger by ALL*, this component system will be taken as a AND task. If the trigger method of a component system is *trigger by SOME*, this component system will be considered as an OR task. In addition, the latency of interaction between two specific tasks can be derived from the constraint of latency imposed on corresponding interaction described in the internal view computational model. According to precedence relationships, we can compute the start time for each task described in the precedence graph. The principle for this computation is described as follows:

If T_x is an AND task, $T_y \prec T_x$ and $T_z \prec T_x$,

$$T_x.starttime \geq MAX\,(T_x.readytime\,, MAX\,(T_y.starttime + T_y.met + I_{yx}.lat\,, T_z.starttime + T_z.met + I_{zx}.lat\,))$$

If T_x is an OR task, $T_y \prec T_x$ and $T_z \prec T_x$,

$$T_x.starttime \geq MAX\,(T_x.readytime\,, MIN\,(T_y.starttime + T_y.met + I_{yx}.lat\,, T_z.starttime + T_z.met + I_{zx}.lat\,))$$

B.3 Dynamic Scheduling Algorithm for EoS

In this section, we will extend the SoES dynamic schedule analysis to implement the dynamic schedule analysis for the EoS.

Assume there are m processors $(m > 1)$, denoted as $p_1, p_2, ..., p_m$. Each processor p_i $(i = 1, ..., m)$ is assigned a real number t_i $(i = 1, ..., m)$, which is proportional to its speed, i.e., faster processor is assigned to a greater t_i. At the same time, $\exists i$, $\exists j$, where $1 \leq i \leq m$, $1 \leq j \leq m$, $i \neq j$, and $t_i \neq t_j$.

B.3.1 Definitions for the EoS Dynamic Scheduling Algorithm

In this section we will provide the definitions used for the EOS dynamic scheduling algorithm.

Definition B.3.1.1: A task is feasible if its timing constraint and resource requirements are met in the schedule. A schedule for a set is said to be a feasible schedule if all the tasks are feasible in the schedule.

Definition B.3.1.2: A partial schedule is a feasible schedule for a subset of tasks. A partial schedule is said to be strongly feasible if all the schedules obtained by extending the current schedule by any one of the remaining tasks are also feasible.

Definition B.3.1.3: EAT_k^s (or EAT_k^e) is the earliest time when resource R_k becomes available for shared (or exclusive) access.

Definition B.3.1.4: $IEST(T)$ is the ideal earliest start time of task T. Let PE be the set of processors and R_T be the set of resources required by task T. Thus, $IEST(T) = MAX\,(T.starttime, MIN_{p \in PE} availtime(P), MAX_{R_k \in R_T} EAT_k^u)$. Here, $T.starttime$ is the start time for task T. To derive the start time for task T, we should go through the precedence graph for the whole system and find out the precedent-task set of task T. Based on this, we can use the principle described in Section B.2 to compute the start time for task T. $availtime(P)$ denotes the earliest time at which the processor P becomes available for executing a task and

the third term denotes the maximum among the earliest available times of the resources required by task T ($u = s$ for shared mode and $u = e$ for exclusive mode).

Definition B.3.1.5: $avail(T, P)$ denotes the feasibility of task T executing on the processor P. If the deadline of task T can be met by executing on processor P, $avail(T, P) = 1$; otherwise $avail(T, P) = 0$.

Definition B.3.1.6: *sysavailtime* is the minimum available time of processors in whole systems, i.e., $sysavailtime = MIN_{P \in PE}(availtime(P))$.

Definition B.3.1.7: $Spe(P)$ reflects the speed of processor P. The lower the speed of processor P, the larger the value of $Spe(P)$.

Definition B.3.1.8: $Finishtime(P, T)$ denotes the finish time of task T when it executes on processor P.

B.3.2 Overview of the EoS Dynamic Scheduling Algorithm

This section will provide an overview of the scheduling algorithm for both dynamical and integrated scheduling a task sets composed of hard and soft deadlines, plus best effort pseudo deadlines. The algorithm is based on heuristic searching. When a set of new component systems with precedence and resource constraints arrive at an EoS, the tasks associated with these component systems along with other unscheduled tasks already in an EoS will trigger this scheduling algorithm.

In this algorithm, the schedule starts at the root of the search tree, which is an empty schedule. The algorithm tries to extend the schedule (with one or more tasks) by moving to one of the vertices at the next level in the search tree until a fully feasible schedule is derived. For this purpose, we set the feasibility check window with K size for the unscheduled task set. We will then check the feasibility of the current schedule by extending the schedule with each task in this window. Once the current schedule is not feasible due to a certain task in the feasibility check window, we will use a degradation policy described in Section B.3.4to degrade this task.

If the current schedule is strongly feasible, we will choose the task with the smallest value of the heuristic function H to extend the current schedule. The heuristic function H for task selection is $H(T) = D_T + W * IEST(T)$, where D_T is the deadline of task T and W is a weight value. Once a task, within the feasibility check window, is selected to extend the current schedule, the task will be assigned to a specific processor according to the task assignment policy, which is described in Section B.3.3.

Otherwise, if the current schedule is not strongly feasible (even after all soft real-time tasks in the feasibility check window that caused the infeasibility of the current schedule have been degraded to the lowest service level), we will back track to the previous search level. We will then extend the current schedule by another task having the next smallest H value in this search level and choose a suitable processor for this task according to the same task assignment policy.

Furthermore, if a task in the feasibility check window is associated with a new component system and is found to result in infeasibility of the current schedule, we will compute and the maximum execution time threshold (METT) for this task. The computed value for the METT is the upper bound of the maximum execution time that this task needs to meet to make the extended current schedule feasible. Assuming that E_T^* represents the METT of task T, then $E_T^* = D_T - IEST(T)$.

After the feasibility check for the current schedule, the feasibility check window is moved by one task. The above process is repeated until a complete feasible schedule is found or no more backtracking is possible. If a complete feasible schedule is found, then adding the new component systems will not violate the schedulability of the whole system so that the dynamic reconfiguration is accepted. Otherwise, the dynamic reconfiguration is rejected since adding the new component systems will damage the high confidence of the whole system.

In the case where the dynamic reconfiguration is rejected, we need to further detect if the unschedulability of the whole system resulted from certain new component systems. If so, the recorded METT of tasks associated with these new component systems will be provided as the suggested maximum execution times to the "training process" for virtually reducing their maximum execution times.

B.3.3 Task Assignment Policy for the EoS

The task assignment policy is used to select the most suitable processor to execute task T. Task T is selected to extend the current partial schedule during heuristic searching. We will use a heuristic function S to achieve this goal. This heuristic function was developed by Y. Qiao, et al..[8] For each processor P, $S(P) = availtime(P) + Spe(P)$. The basic idea of the task assignment policy is to select the processor for executing task T based on the largest heuristic function S value for a given processor P. Since earlier *sysavailtime* and faster speed of the processor having the minimum available time after executing task T can lead to higher feasibility of unscheduled tasks. Furthermore, we will check whether all successive tasks of task T can meet their deadlines if T is executed in the processor having the largest S value. If the answer is negative, we should choose the processor having the next larger S value.

In this dynamic scheduling analysis algorithm, the new task assignment policy is denoted as the function $choosep(T)$. This function returns the identification number of the processor selected for executing task T, otherwise NULL is returned. The detailed task assignment policy is listed below:

1) If the resources required by task T are no more than processors, then choose the processor P that can meet the following constraint for task T,.
2) If task T requires some other resources besides the processors, then employ the method listed below:
   2.1) If there is no intersection between the resources required by task T and the resources required by the remaining tasks, then employ the same processor selections policy as 1) for task T.

2.2) If there is an intersection between the resources required by task T and the resources required by the remaining tasks, and both task T and the remaining tasks have shared access to the resources in this intersection, then again employ the same processor selections policy as 1) for task T.

2.3) Otherwise, the policy used to choose the processor for task T is listed below: (We assume that processor P' meets the following constraints:
$availtime(P') = MAX_{pe\in\{pe|pe\in PE \quad and \quad avail(T,pe)==1\}}(availtime(pe))$)

2.3.1) If $r_T \le ESTR_T$ and $ESTR_T = MAX_{pe\in\{pe|pe\in PE \quad and \quad avail(T,pe)==1\}}(availtime(pe))$ and $S(P') == MAX_{pe\in\{pe|pe\in PE \quad and \quad avail(T,pe)==1\}}S(pe)$, we will employ the same processor selections policy as 1) for task T. r_T is the time resources are available for task T and $ESTR_T$ is the earliest available time of all resources required by task T.

2.3.2) If $r_T \ge ESTR_T$ and $ESTR_T \ge MAX_{pe\in\{pe|pe\in PE \quad and \quad avail(T,pe)==1\}}(availtime(pe))$ and $Spe(P') == MAX_{pe\in\{pe|pe\in PE \quad and \quad avail(T,pe)==1\}}(Spe(pe))$, we will employ the same processor selections policy as 1) for task T.

2.3.3) Otherwise, we will choose the processor P that can meet the following constraint for task T:
$Finishtime(P,T) = MIN_{pe\in\{pe|pe\in PE \quad and \quad avail(T,pe)==1\}}(Finishtime(pe,T))$

3) Check whether all successive tasks of task T can meet their deadlines under current processor selection.

3.1) If it is not true, choose another processor with the next largest S value for case 1), 2.1), 2.1), 2.3.1), and 2.3.2); choose another processor with the next minimum value of function $Finishtime$ for case 2.3.2).

3.2) Go back to 3) until the processor is selected or no more processors can be selected.

4) If a processor is selected, return the identification number of this processor; otherwise, return $NULL$.

B.3.4 Task Degradation Policy for the EoS

Y. Qiao, et al., developed a degradation policy for SoES that would iteratively degrade the quality of service (QoS) of soft real-time tasks to improve the schedulability of hard real-time tasks. This degradation is done during the feasibility check, where we first use the logic version of the soft real-time task with the longest execution time in the feasibility check window. We assume that the longer the execution time of the logic version, the better the quality of the result. Once the current schedule is not strongly feasible due to unschedulability of a soft real-time task in the feasibility check window, the QoS of this soft real-time task is degraded to the next lower logic version. This degradation will continue until the feasibility check is successful. If the feasibility check is still not successful when the QoS of of this soft real-time task has been degraded to the lowest logic version, a backtracking will occur. We denote the detailed degradation policy for task T with the function *degrade(T)* and the return value is the service level number that can be provide for task T. The service level number corresponds to the logic version of the soft real-time task T. Details of the function *degrade(T) are as follows:*

1) If task T results in infeasibility of the current schedule and task T is a soft real-time task
   1.1) Check the current service level of task T.
   1.2) If the current service level is not the lowest, we will degrade the service level of task T to the next lower one, i.e., select the logic version with the next shorter execution time to the current service level of task T.
   1.3) Repeat 1.1)–1.2) until the service level of task T has been degraded to the lowest one or the feasibility check is successful.
2) Else exit.

B.3.5 Dynamic Scheduling Analysis Algorithm for an Enterprise of Systems

The dynamic scheduling analysis algorithm for an EoS is shown below:

1) Order the task in the task queue in increasing order of their dealines and then start with an empty partial schedule.
2) Check the feasibility window.
3) For $i = 1\ to\ K (or\ less\ K)$
   3.1) If the schedule is not feasible by extending the current schedule with task T_i, then *degrade(T);*
4) Determine whether the current partial schedule is strongly feasible by performing feasibility check for K or less than K tasks in the feasibility check window. If the current partial schedule is strongly feasible, then $feasible = true$; otherwise $feasible = false$.
5) If ($feasible == true$)
   5.1) Compute the heuristic function H value for the K or less than K tasks in the feasibility check windows, where $H(T) = D_T + W * IEST(T)$ for task T.
   5.2) Extend the schedule by task T having the smallest H value in the feasibility check window and choose a suitable process for task T by using the function $choosep(T)$.
   5.3) If ($choosep(T) == NULL$), then go to 5.4).

   Else

   5.4) For each task T in the feasibility check window, if task T is associated with a new component system and is not feasible in the current schedule, compute and record E_T^* (the maximum execution time threshold (METT) of task T)
   5.5) Backtrack to the previous search level.
   5.6) Extend the schedule by the task T' having the next smallest H value in this search level and choose a suitable processor for task T' by using function $choosep(T)$.
6) Move the feasibility check window by one task.
7) Repeat steps 2)–6) until any termination condition listed below is met:
   7.1) A complete feasible schedule has been found.
   7.2) No more backtracking is possible.

8) If a complete feasible schedule is found
   8.1) Accept the addition of the new component system and the reconfiguration.

   Else

   8.2) Reject the additional of the new component system and the reconfiguration.
   8.3) If the task associated with the new component system results in unschedulability of the whole system, provide the METT of the task to the "training process."

---

[1] Joint Doctrine for Military Operations Other Than War, Joint Pub 3-07, 16 June 1995.
[2] 2004 Indian Ocean Earthquake, Wikipedia, http://en.wikipedia.org/wiki/2004_Indian_Ocean_earthquake, Accessed on 12 November 2006.
[3] Hurricane Katrina, Wikipedia, http://en.wikipedia.org/wiki/Hurricane_Katrina, Accessed on 12 November 2006.
[4] Qiao, Y., H. Wang, Luqi, and V. Berzins, "An Admission Control Method for Dynamic Software Reconfiguration in Complex Embedded Systems," *International Journal of Computers and Their Applications*, Vol. 13, No. 1, March, 2006, pp. 28-38.
[5] C4ISR for Future Naval Strike Groups, Committee on C4ISR for Future Naval Strike Groups, National Research Council, 2006, http://www.nap.edu/catalog/11605.html, Accessed on 13 November 2006.
[6] Ibid.
[7] Ibid.
[8] Qiao, Y., H. Wang, Luqi, and V. Berzins, "An Admission Control Method for Dynamic Software Reconfiguration in Complex Embedded Systems," *International Journal of Computers and Their Applications*, Vol. 13, No. 1, March, 2006, pp. 28-3 8.

Use of a Systems Information Broker to Aide in the Dynamic Interfacing of C2 Nodes

Dagohoy H. Anunciado

12th ICCRTS
Adapting C2 to the 21st Century
19-21 June 2007
Naval War College
Newport, RI

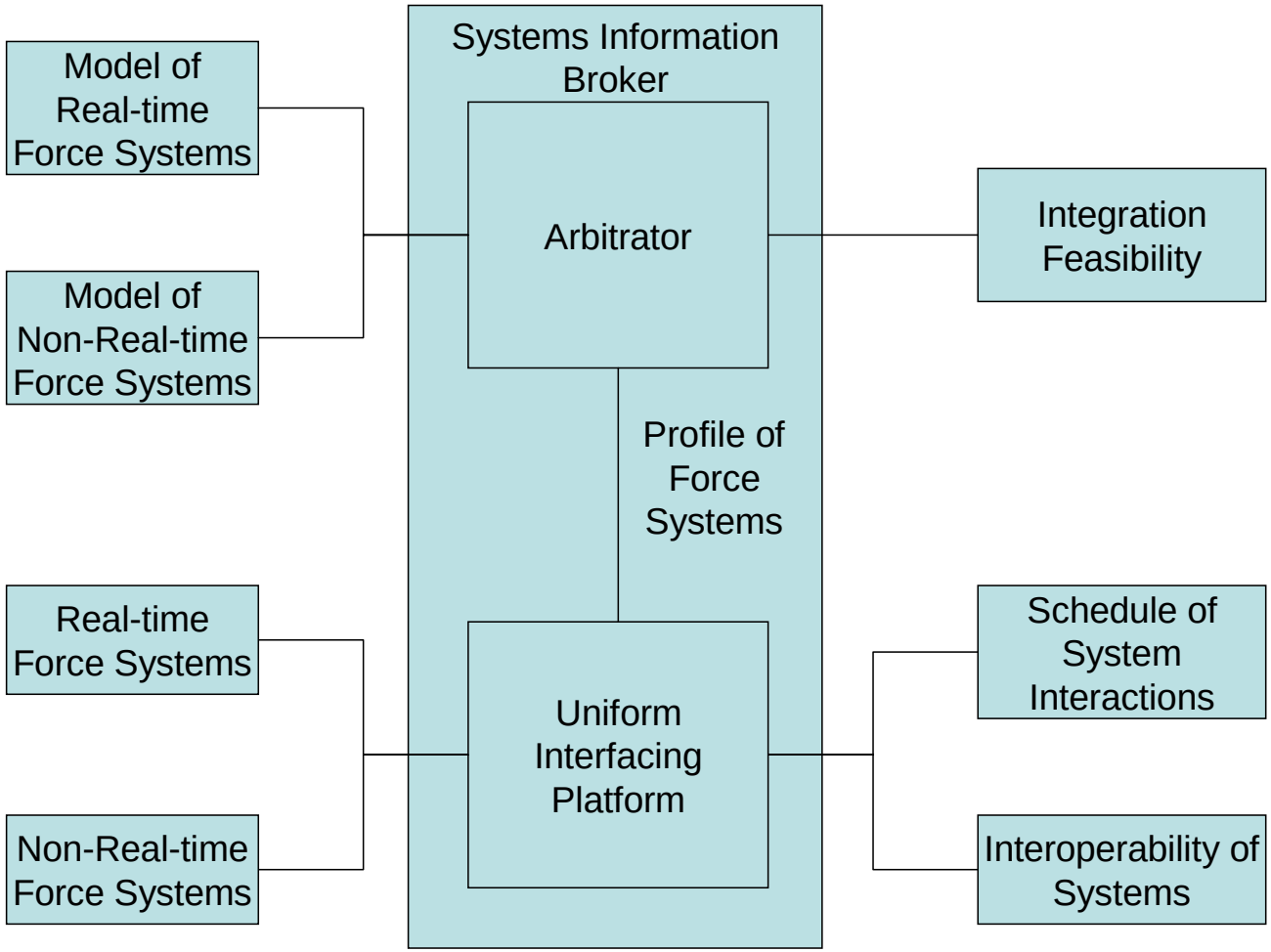
Introduction

- Changing World Events Require Changes in Missions for Military Forces
- Accomplishing Mission Requires Resources that C2 Nodes Provide
- Method for Interfacing C2 Nodes into a Enterprise of Systems (EoS) that are Used to Accomplish Mission

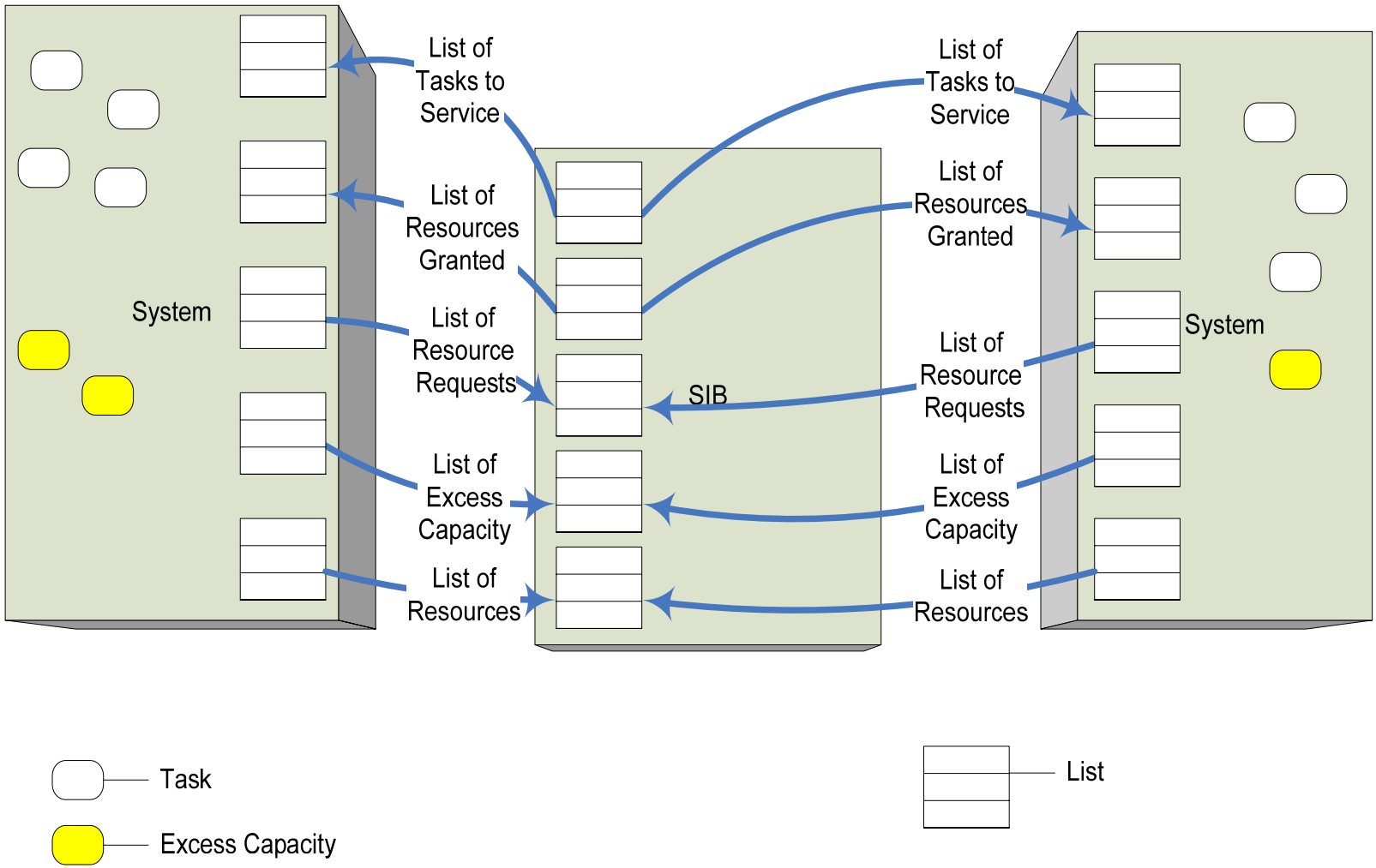
Example World Events

- Indian Ocean Tsunami
- Hurricane Katrina

Systems Information Broker (SIB) Architecture and Framework



SIB Constructs



Admission Control Method for EoS

- Two views
  - External View – functional aspects of EoS
  - Internal View – non-functional aspect of EoS
- Dynamic Scheduling Analysis for EoS
  - Dynamic Scheduling Task Model
  - Dynamic Scheduling Precedence Graph
  - Dynamic Scheduling Algorithm
  - Task Degradation Policy
  - Dynamic Scheduling Analysis Algorithm

Modeling Constructs for Interfacing Systems

- System Information Base (SIB)
- Systems
  - Systems in the EoS
  - Systems requesting admission to the EoS
- List of Tasks to Service
- List of Resources Requested
- List of Resources Granted (May be Remote Resources)
- List of Excess Capacity
- List of Resource (Local System Resources)

Assumptions

- Indivdual systems designed to handle normal local tasks and task loads
- A system’s excess capacity is available for remote systems

Real-time System Resources Modeled

Resource	Resource Concern
System	
CPU	Lack of CPU cycles to complete a task calculation that will cause a task to miss its deadline.
Memory	Lack of memory causing a task to miss its deadline.
I/O	Waiting for I/O resources that causes a task to miss its deadline.
Network	
Bandwidth	Lack of Bandwidth that causes a task to miss its deadline.
Quality of Service	Jitter and Latency that degrade information flow and causes a task to miss its deadline.

Non-real-time Systems Resources Modeled

Resource	Resource Concern
System	
CPU	Lack of CPU cycles prevents a task from completing its computations in a usefully timeframe.
Memory	Lack of memory prevents a task from completing its computations in a usefully timeframe.
I/O	Waiting for I/O resources prevents a task from completing its computations in a usefully timeframe..
Network	
Bandwidth	Lack of Bandwidth case prevents a task from completing its computations in a usefully timeframe.
Quality of Service	Jitter and Latency that degrade information flow and prevents a task from completing its computations in a usefully timeframe.

Conclusion

- Admission Control for an Enterprise of Systems Provides a Method for Interfacing C2 Nodes
- Interfaced C2 Nodes Provide Resources
- C2 resources Provide Means to Accomplish a Mission